Faculdade de Engenharia da Universidade do Porto

Ship Escape

LCOM Final Project – Class 1 – Group 2

Made by:

João André Silva Roleira Marinho, up201905952

Table of Contents

Introduction

The final project consists of a 2D game, Ship Escape, which is in a Top-Down view, although the idea of depth created by its background.

The player, an astronaut, who is trying to save his crew mates from the space invaders, has 300 seconds to complete all the tasks distributed around the map while avoiding getting killed by these creatures. If the time expires the player loses the game and can restart it as many times as he wants.

In order to finish the tasks, he may use the mouse and/or the keyboard, he may as well kill the opponents with his projectiles, but careful you can only shoot again when the projectile disappears.

1. User instructions

1.1 Main Menu

As soon as you start the game, the following menu is displayed:

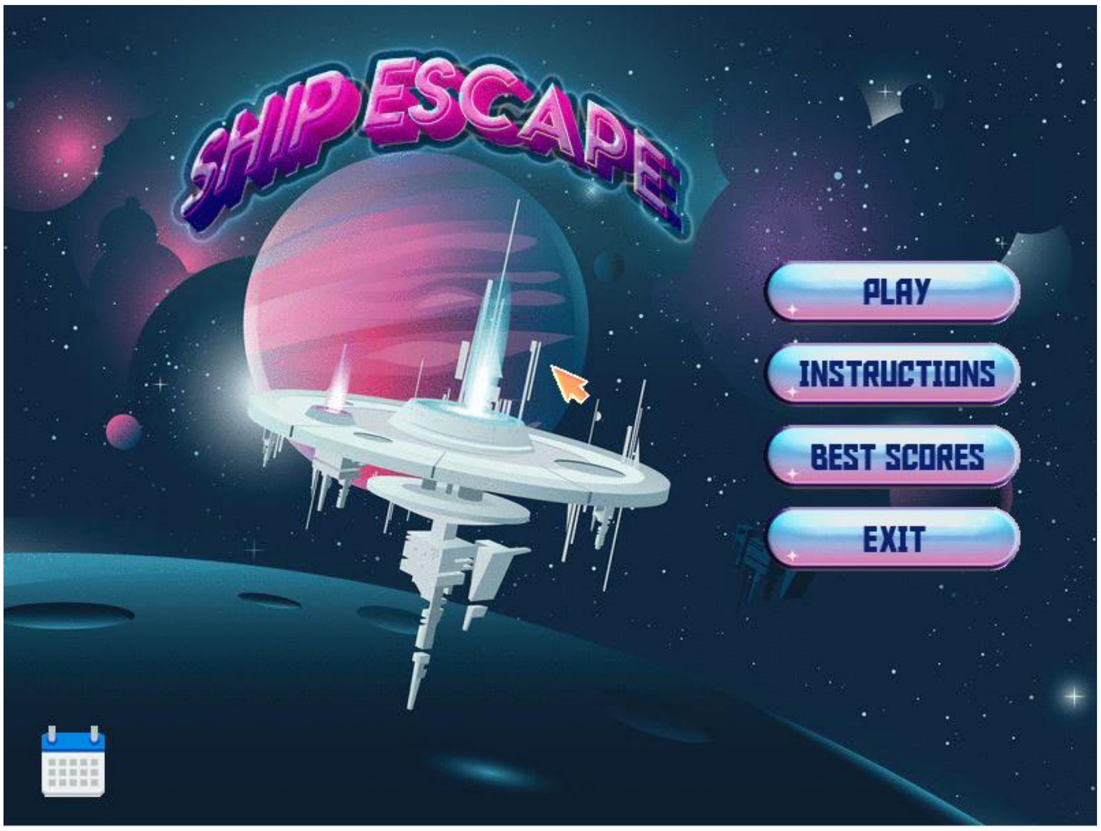


Figure 1. Main menu image

There are four possible options: PLAY, INSTRUCTIONS, BEST SCORES and EXIT.

With the movement of the mouse, you can control the cursor that you see on the screen and by clicking with the left mouse button you can choose one of the options stated before.

1.1.1 Date

By hovering with the cursor over the calendar in the left bottom corner the current date will be displayed, and the calendar will change colour:

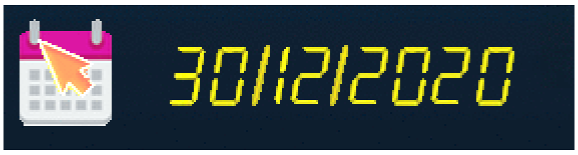


Figure 2 Calendar with date overview

1.1.2 Exit

If you select the EXIT option, the program will shut down (end the game).

Figure 3 Exit button image

1.2 Play Menu

By pressing the PLAY option,

Figure 4 Play button image

the real game begins, the player starts in what is called the CAFETERIA room in the centre of the screen. The countdown time is shown and starts decreasing right away.

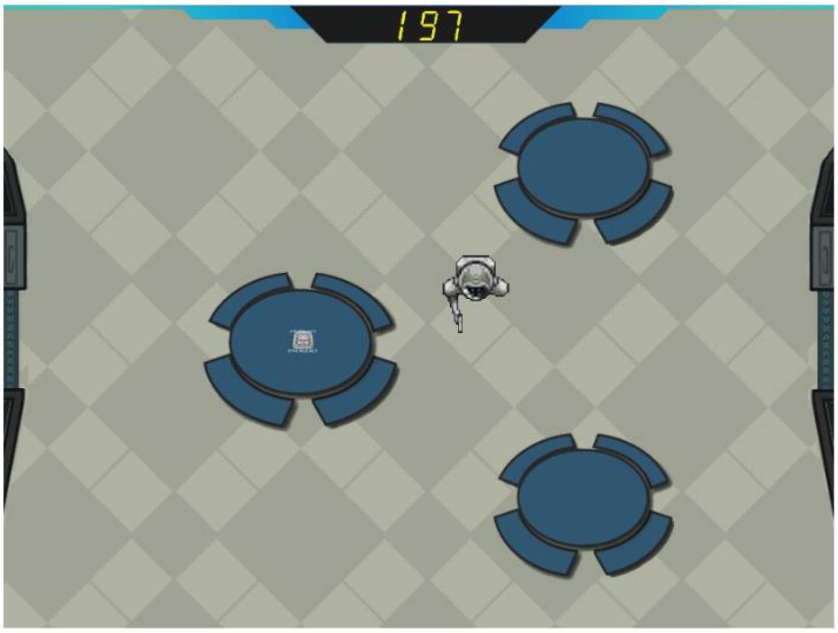


Figure 5 Cafeteria room image

The player movement is controlled with the keyboard, both with the W, A, S, D keys or with the up, left, down, right arrow keys.

The player may travel around the map passing through the doors and existing hallways.

1.2.1 Game map

Concerning the orientation of the player throughout the game there exists a game map menu that can be accessed with the M key in the keyboard. There, the player obtains information on which room he must go for the remaining tasks, but he may need to be fast because the clock never stops. If he wants to leave this menu both the ESC and M key may be used.

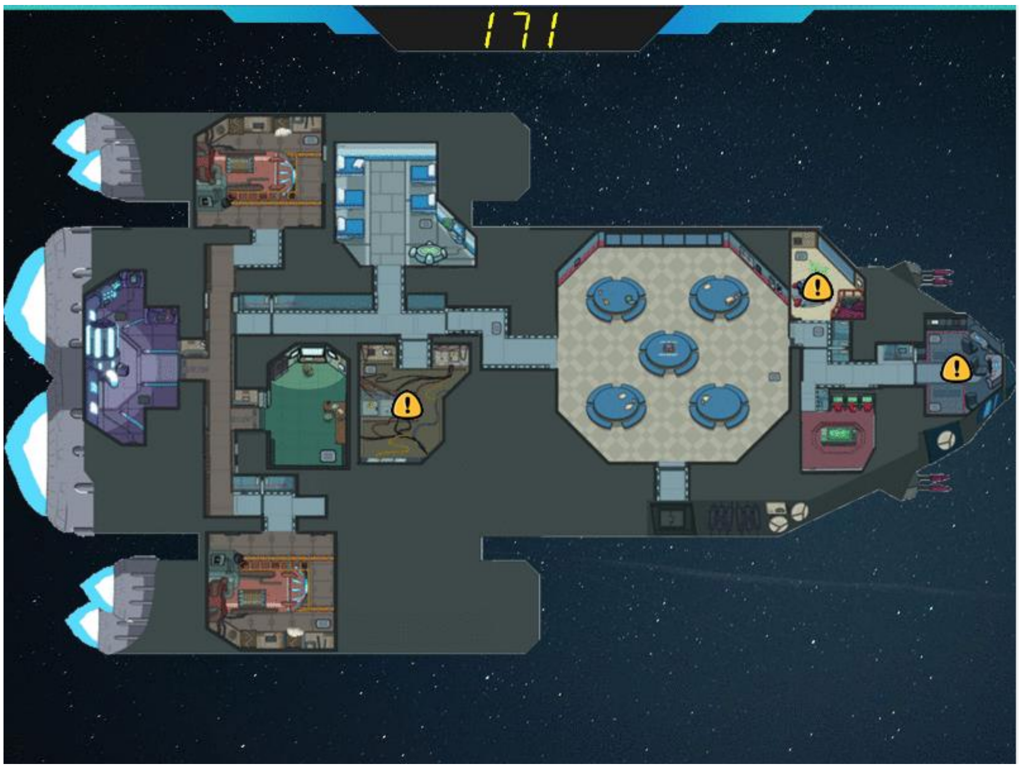


Figure 6 Game map image

1.2.2 Pause menu

If the player needs a break or wants to go back to the main menu, he may do it by pressing the ESC key in the playing mode. This will load the pause menu, where the countdown stops, and two options are available: RESUME and MENU.

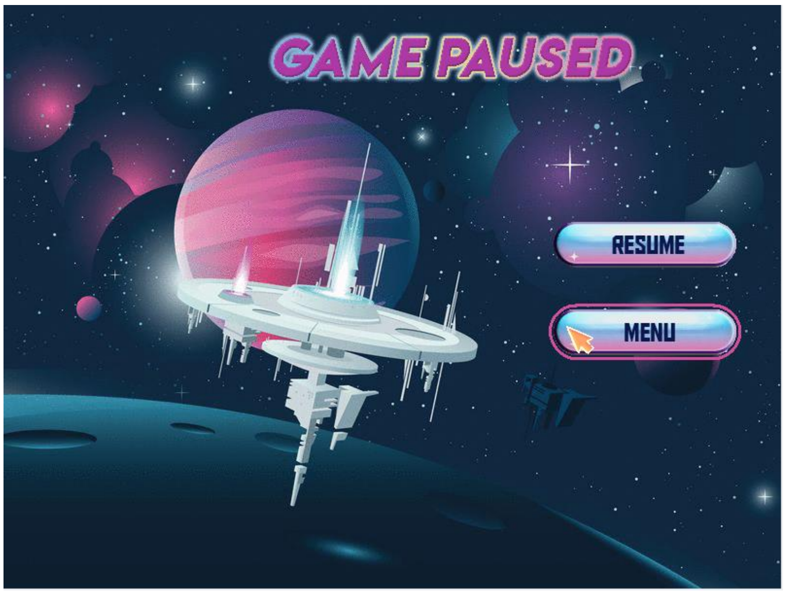


Figure 7 Pause menu image

If selected, the resume button will take the player back to the previous state of the game. On the other hand, the menu button loads the main menu once again, where he can restart playing the game from the very beginning.

1.2.3 Opponents

Throughout the map are distributed opponents:

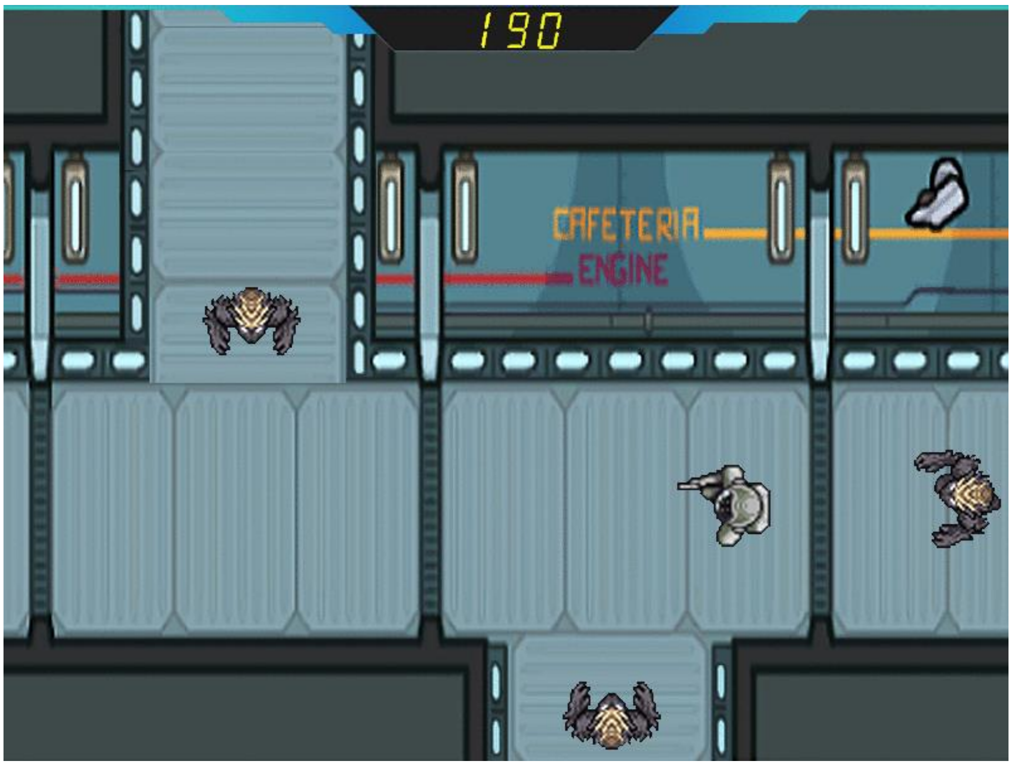


Figure 8 Hallway3 image with opponents

These little creatures will make the player's mission a struggle. In order to survive the player must dodge the monsters, or with the help of is gun he may shoot his projectiles towards them using the SPACE BAR key in the keyboard, otherwise they will attack and kill him. Once the player is dead the defeat menu will be loaded.

So that the game becomes more fun, from time to time the fallen invaders will come back to life and try to kill the adventurer.

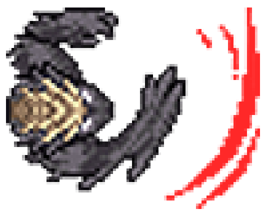


Figure 9 Opponent attack final animation

1.2.4 Tasks

The main objective of the player, said previously, is to complete all the tasks before he runs out of time. Their locations are shown in the game map and appear in the room pointed by a yellow arrow.

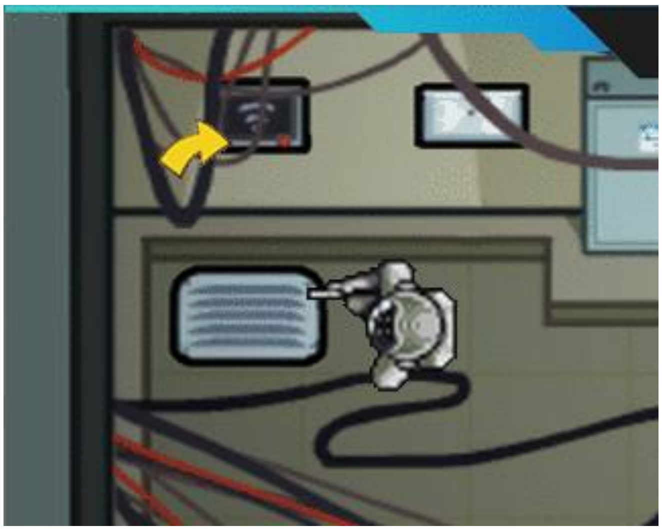


Figure 10 Arrow pointing to task image

There are different types of tasks, the ice task, the ship task, the download task, and others. The player may activate a task by pressing the E key in the keyboard, if he is close enough to it, the task will appear on the screen.

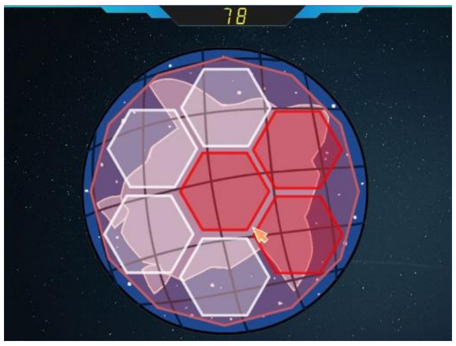


Figure 11 Ice task image

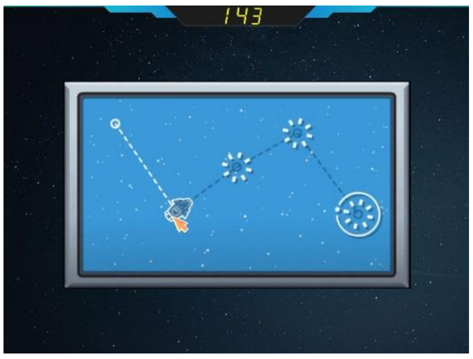


Figure 12 Ship task image

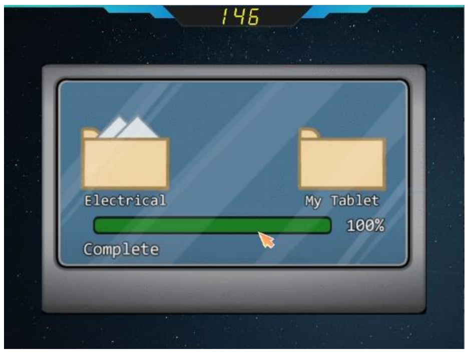


Figure 13 Download task image

1.2.5 Victory/Defeat

The player either wins or loses the game, if he wins, the victory screen will be loaded showing the message of success and the points collected with the run, the faster the player finishes the tasks the more points he gets. If he gets a score worthy of the top 5 scores it will be available to check in the Best Scores menu.

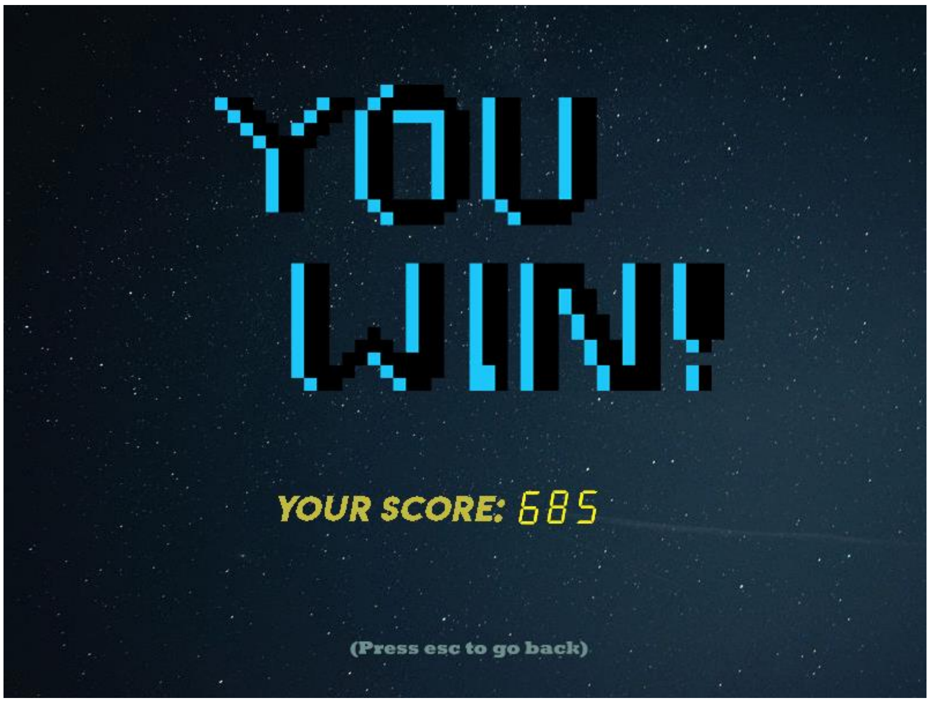


Figure 14 Victory screen image

Moreover, if the player loses, the defeat screen will be loaded. In both menus the player must use the ESC key to retrieve to the main menu.

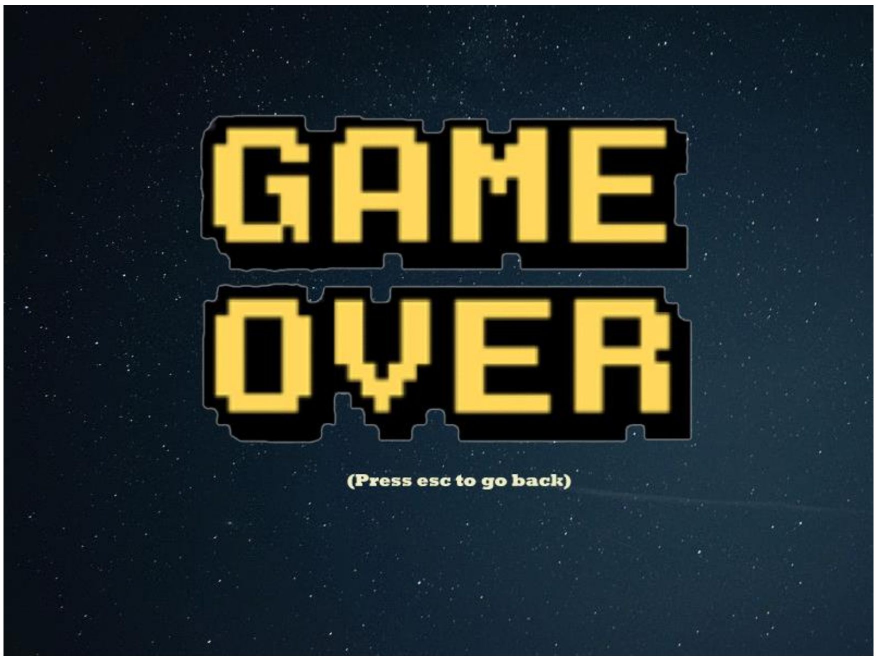


Figure 15 Defeat screen image

1.2.6 Rooms

The different locations the map is made of are the rooms. Each room has a different number of opponents and tasks. Some examples are the cafeteria, navigation, electrical, weapons, security, reactor as well as 3 hallways and others.

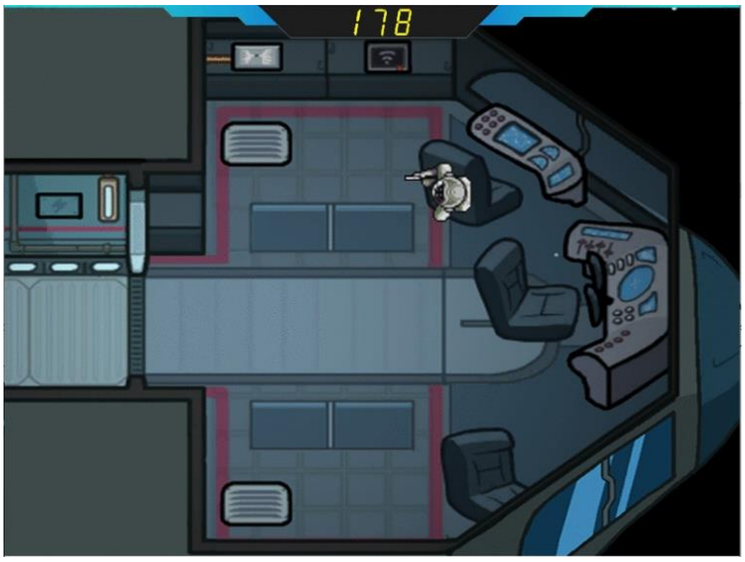


Figure 16 Navigation room image

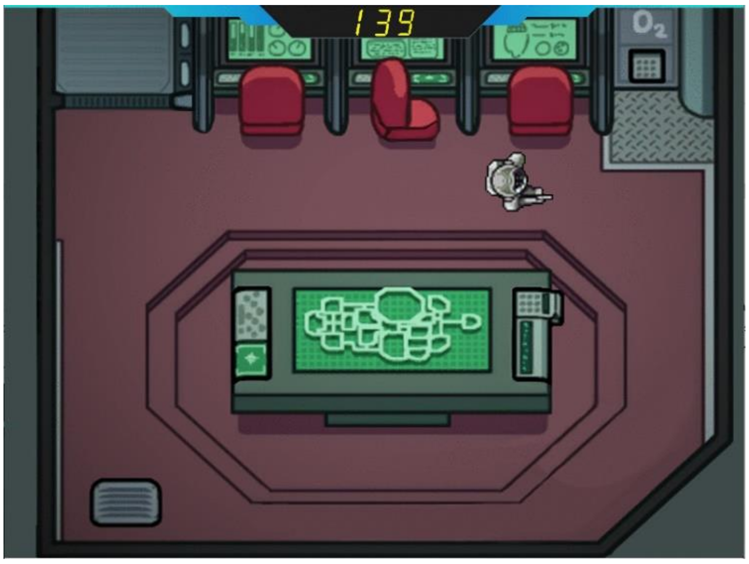
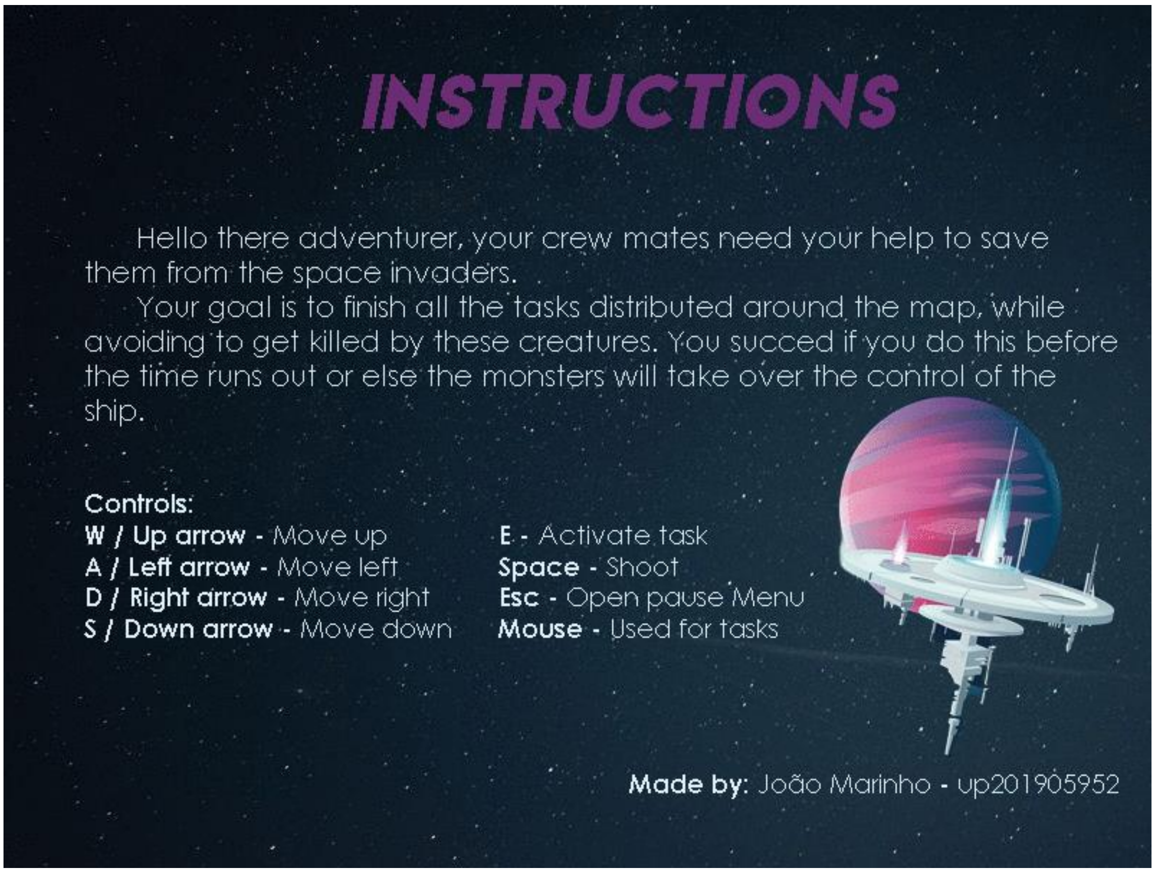


Figure 17 Admin room image

1.3 Instructions menu

In the main menu after the player selects the INSTRUCTIONS button, the game objective and instructions will be displayed on the screen as the following:



INSTRUCTIONS

Hello there adventurer, your crew mates need your help to save them from the space invaders.

Your goal is to finish all the tasks distributed around the map, while avoiding to get killed by these creatures. You succed if you do this before the time runs out or else the monsters will take over the control of the ship.

Controls:

W / Up arrow - Move up
A / Left arrow - Move left
D / Right arrow - Move right
S / Down arrow - Move down

E - Activate task
Space - Shoot
Esc - Open pause Menu
Mouse - Used for tasks

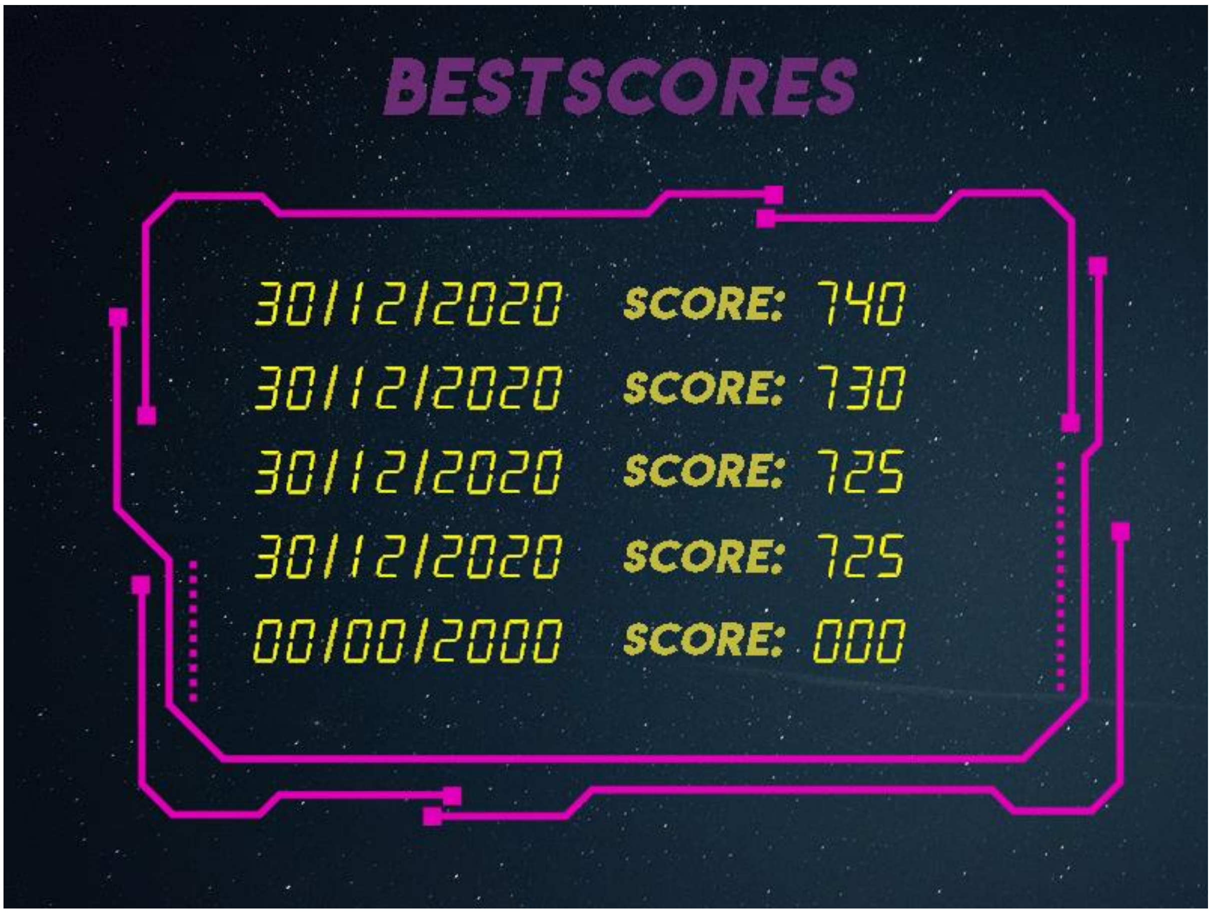
Made by: João Marinho - up201905952

Figure 18 Instructions menu image

In order to leave this menu, the player must press the ESC key.

1.4 Best Scores menu

The best scores menu, saves the 5 best scores achieved in the program run with the respective date provided by the RTC interrupts. These are displayed by descending order in terms of points gained.



30/12/2020	SCORE:	740
30/12/2020	SCORE:	730
30/12/2020	SCORE:	725
30/12/2020	SCORE:	725
00/00/2000	SCORE:	000

Figure 19 Best Scores menu image

In order to leave this menu, the player must press the ESC key.

2. Project status

Device	Purpose	Interrupts
Video card	Display the game and all its menus	N / A
Keyboard	Player movement, access the map, activate tasks, and shoot projectiles	Yes
Mouse	Choose menu options and control tasks	Yes
Timer	Controls the frame rate, and the characters animations as well as the count down time.	Yes
RTC	Display the current date	Yes

2.1 Video card

Mode	Screen Resolution	Colour Mode	Number of Colours
0x115	800x600 pixels	(8:8:8) 24 bits per pixel (direct mode)	16.8M (2^24)

In this project the Double Buffering technique was used to help the game transitions and the animation of all characters. This approach consists of having a backend buffer in which all the information is copied to the main one after a timer interrupt with the function copy_to_vram.

Many are the animations present in the game, the movement of the player and the opponents, the collision of the projectiles, the attack of the opponents and each task animations.

The player has a sequence of 8 animations, all facing the same way (right), these change according to the timer ticks and if the player is moving or not. The player movement is obtained by erasing his last position and drawing him again in the new one (x_0+xspeed if moving horizontally and the same for y), this way there is no need to go through all the resolution of the screen in other to draw a rather small object.

To draw the player as well as all other objects that have a direction related to them, facing other directions, their pixmap are read in different ways so that they are looking the direction desired.

Aside from the movement there are other animations such as the attack of the opponent and the explosion of the projectile, that work similarly to the ones referred previously.

The collision between the player, opponent or projectiles with the room obstacles is made with the help of a xpm that has in white (transparent) the area which the objects can move, and not white (coloured) the area where the obstacles are located. Thus, the verification is made pixel by pixel when the object is moving.

The collision between the player with the opponents and the projectiles, are verified by their next timer interrupt position, if an opponent is hit with a projectile he dies and disappears, if an opponent attacks a player, the last one only disappears when the slash of the opponent is done.

All the images in the game use xpms. Most edited in photoshop and downloaded from the web, with free usage.

The main functions related to the video card are in the **Videocard.c** file.

2.2 Keyboard

The keyboard is mainly used to control the movement of the player and access certain menus. By pressing the W, A, S, D or the up, left, right, down arrow keys, the player moves accordingly. You may as well use the SPACE key to shoot a projectile, and the E key to activate a task if close enough to it, verified with the function near_task.

The ESC key has different purposes according to the menu the player is in, if the current menu is the INSTRUCTIONS, BESTSCORES, VICTORY, DEFEAT or even in a TASK, the player will retrieve to the menu before or to the MAIN. If the player is in play mode, pressing the ESC key will open the PAUSE menu.

There is also a game map that can be accessed with the M key, in play mode.

The main keyboard functions are in the **KeyBoard.c** file, these were mainly developed during the Lab3 week.

2.3 Mouse

The mouse is put in service in the menu as well as in the game. It is represented by a cursor and in both cases, the two applications are used: the buttons and the displacement or the mouse.

In the menu it is mainly used to select the different options, and to show the date in the MAIN menu if hovered over the calendar. In the game, it is used in the TASK menus and has different purposes conform the task the player is in. With the function get_mouse_event the information about the event of the mouse is processed and saved in a more friendly manner.

There are other functions such as the ship_gesture_handler that manage the information received from the mouse in order to check if a certain movement was made.

The main mouse functions are in the **mouse.c** file, some were developed during the Lab4 week.

2.4 Timer

The timer’s main function is to update the game screen, more precisely 60 times per second. This means that when a timer interrupt is received the player’s animation, opponents and projectiles are updated, the collisions between these three components and the room are managed by the timer interruptions as well. Mouse collisions between the cursor and the MAIN menu buttons or others, making the button change its image as well as updating the countdown time of the game, it is all a job of the timer.

Thus, it takes a noticeably big part of the project. Its implementation took place during the Lab2 week and some of its functions appear in the **timer.c** file.

2.5 RTC

The RTC's purpose is to display the current date in the MAIN menu. Functions such as the draw_Date and the draw_Number, are used to draw the date on to the screen.

Interrupts are used to update the time, and the function LoadRTC was created to collect all the information needed for the date. To keep things organized a **struct Date** was also implemented saving the current date and the symbol images used for display.

In addition, the RTC is used to show the top 5 best scores during that program run in the BESTSCORES menu.

All its functions are located in the **Rtc.c** file.

3. Code organization/structure

3.1 I/O Device Modules

3.1.1 Timer Modules

Timer.c

Overall Weight: 6%

Description: In this module there are all functions that deal directly with timer interrupts (timer subscribe and unsubscribe). Its development was made during the lab2 week.

I8254.h

Overall Weight: N/A

Description: In this module all the significant constant variables for the manipulation of the timer device are present. Those were given to students during the development of lab2.

3.1.2 Keyboard Modules

KeyBoard.c

Overall Weight: 5%

Description: In this module there are all functions that deal directly with keyboard interrupts (keyboard subscribe, unsubscribe, and interrupt handler). Its development was made during the lab3 week.

I8042.h

Overall Weight: N/A

Description: In this module all the significant constant variables for the manipulation of the keyboard and the mouse are present (both use the KBC Controller). Its development was made during the lab3 and lab4 week, with some adjustments for the project.

3.1.3 Mouse Modules

Mouse.c

Overall Weight: 5%

Description: In this module there are all functions that deal directly with mouse interrupts (mouse subscribe, unsubscribe, interrupt handler and others). Its development was made during the lab4 week.

Furthermore, a **struct Cursor** was used to save the menu and tasks cursor information (position and image).

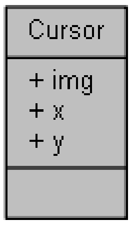


Figure 20 Cursor struct diagram

3.1.4 Video Card Modules

Videocard.c

Overall Weight: 6%

Description: In this module there are all functions that deal with the video card (initialization, correct mode setup, allocation of memory, paint pixels and copy from the second buffer to the main one). Its development was made during the lab5 week with the addition of the double buffering technique.

An **enum** typedef was also created, the **Direction**, which is used to know the direction of certain objects.

VideocardConstants.h

Overall Weight: N/A

Description: In this module, as the name implies, all the significant constant variables for the manipulation of the video card device are present. Its development was made during the lab5 week.

3.1.5 RTC Modules

Rtc.c

Overall Weight: 5%

Description: In this module there are all functions that deal directly with RTC interrupts (RTC subscribe, unsubscribe, read registers and others).

Furthermore, a **struct Date** was used to save the current date and symbols for its display.

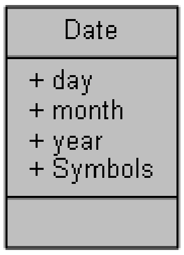


Figure 21 Date struct diagram

RtcConstants.h

Overall Weight: N/A

Description: In this module, as the name implies, all the significant constant variables for the manipulation of the RTC device are present.

3.2 Game Modules

3.2.1 Game Module

Game.c

Overall Weight: 15%

Description: In this module its where the main game loop is located, which receives and handles consequent interruptions from the different devices that able the play of the game. A state machine was implemented in order to call the right interrupt handler, according to the state of the game, with the receiveInterrupt function.

In addition, the Play interrupt handler is present in this module, it is a function that handles the interrupts while in the PLAY mode, it controls the player, the projectiles, the opponents, the game countdown, basically all of the things that are being shown in this mode.

Two other functions were implemented in this module, the roomTransition one and the ResetGame.

The first is used to check if the player is changing rooms, returning true if so and false otherwise.

The second is used to reset the game back to its initial state, resets the opponents, the tasks, the player, the game counter as well as the room.

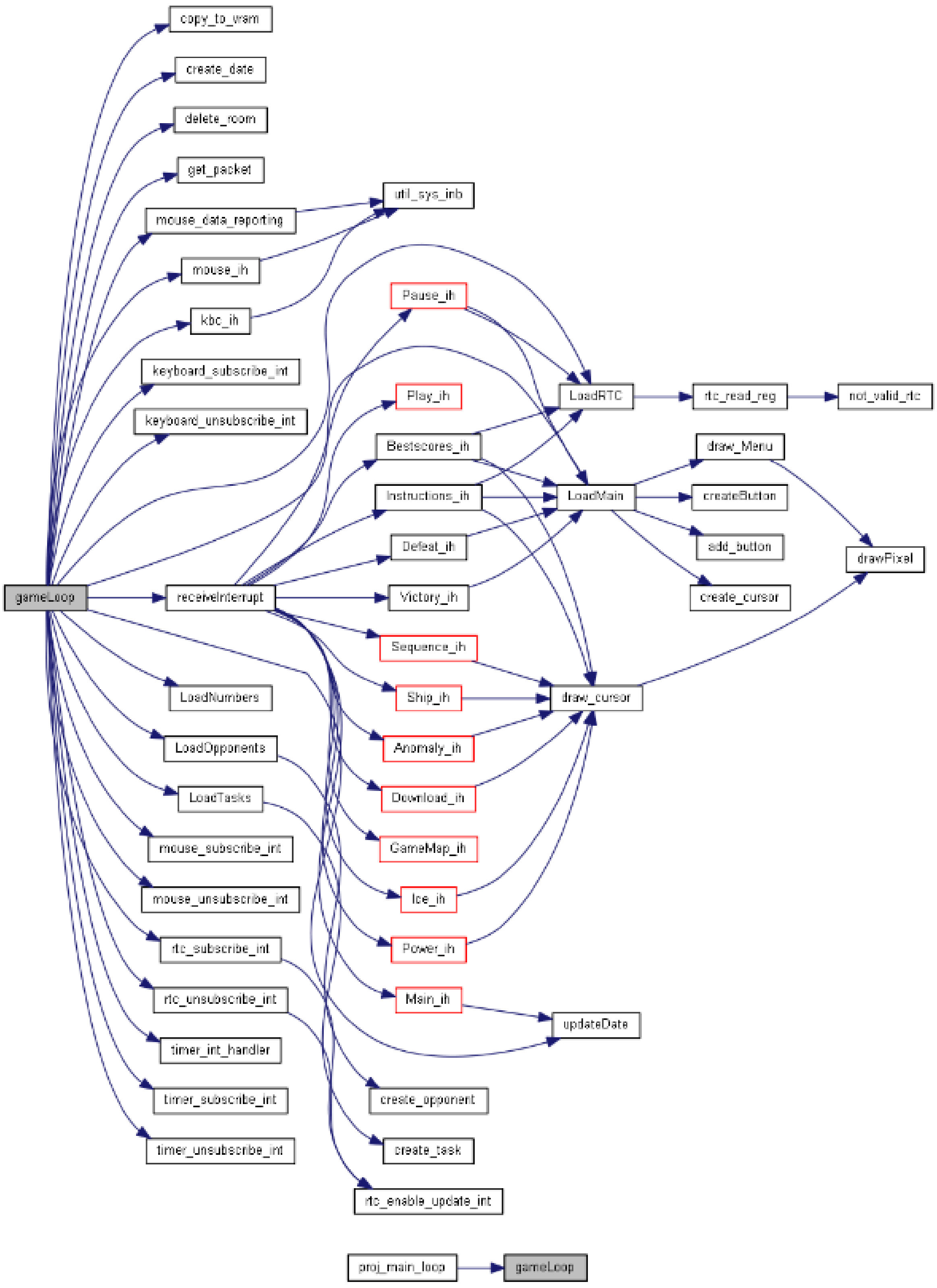


Figura 22 gameLoop caller graph

3.2.2 Menu Module

Menus.c

Overall Weight: 15%

Description: The code presented in this module is related to all the different menus, aside from the PLAY one. It is an especially important module due to its general purpose.

All the menu interrupt handlers as well as their load function are available here.

Furthermore, some task gesture handlers, the management of the top 5 scores, their display and the Date are all things the Menu module takes care of.

Some data structures here implemented are the **GameTimer** and **Score stucts**.

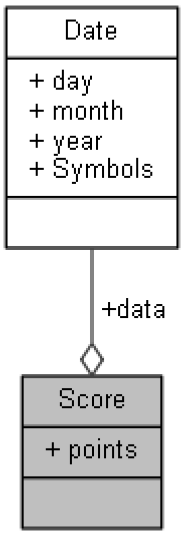


Figure 22 Score struct diagram

Score: Saves information on a successful run by the player, points obtained and date it happened.

GameTimer: Saves the number images (1,2,3 ...) in other to display the countdown time and other objects that use numbers.

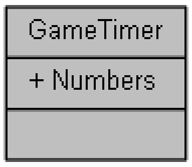


Figure 23 GameTimer struct diagram

3.2.3 Buttons Module

Buttons.c

Overall Weight: 4%

Description: This module's main subject is the **Button** data structure.

This struct represents all the buttons in the MAIN and PAUSE menu. It saves information on the type of button, x, and y position, if the cursor is over or not and the different images of the button.

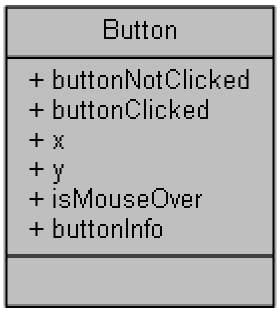


Figure 24 Button struct diagram

3.2.4 Rooms Module

Rooms.c

Overall Weight: 7%

Description: This module concerns the different game rooms (CAFETERIA, ADMIN, NAVIGATION, ELETRICAL and a lot more), its functions allow to load the current room, draw it, and delete it (free the memory allocated).

For this is used a **struct** called **Room**, a single global variable exists with this type, and it has a roomType (enum with different room names), the room background and the respective obstacles.

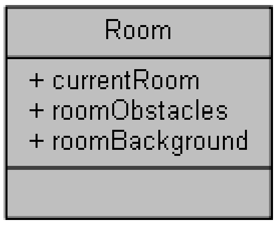


Figure 25 Room struct diagram

3.2.5 Tasks Module

Tasks.c

Overall Weight: 8%

Description: This module concerns the different game tasks (ICE, SHIP, DOWNLOAD, and others).

The functions present in the module allow to create new tasks, load, and finish them by changing the bool **isFinished** in the **Task struct** to true**.**

Other functions in charge of drawing the warnings in the MAP menu, and drawing the arrows pointing to the task in the respective room, were also implemented here.

In order to find if a player is near a task, the near_task function was created as well.

The **Task struct** saves the position, the respective room, all the animation images and a boolean value used to know if a task as been completed or not. In this project an array of **Task** pointers is used to save all the game tasks.

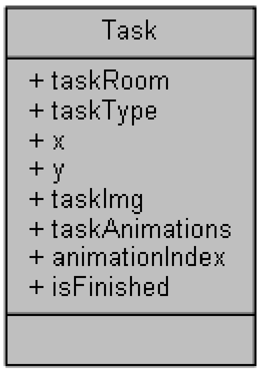


Figure 26 Task struct diagram

3.2.6 Player Module

Player.c

Overall Weight: 10%

Description: The code presented in this module is related to the player object in the game.

In this file were implemented functions with the purpose of controlling the player movement, animations, draw and removal of screen.

To check its collision with the room obstacles and the room opponents, two functions were created: room_player_collision and player_opponent_collision.

All this information was saved in the **Player struct**; a single global variable exists with this type, which contains the player x and y position, direction, animation images, speed in both axis and a boolean that tells if he is alive or not.

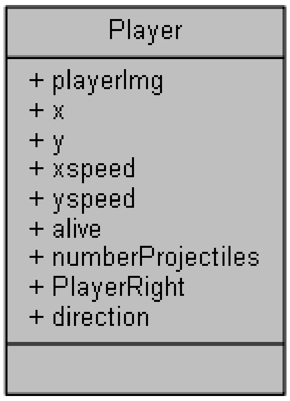


Figure 27 Player struct diagram

3.2.7 Projectile Module

Projectile.c

Overall Weight: 5%

Description: This module is relative to the player projectile. Here were created functions that allow the player to shoot a projectile, draw and erase it from the screen, animate the explosion when it collides with an object and therefore detect the collision both with opponents and room obstacles.

For this being a **struct** called **Projectile** was implemented with the purpose of saving the projectile object position, speed, direction, animation images and a bool that tells if the projectile is being shown or not.

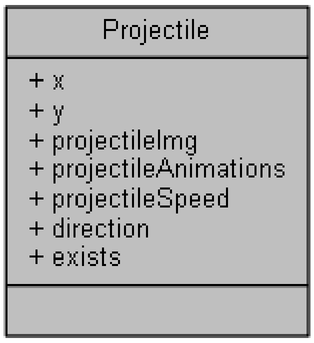


Figure 28 Projectile struct diagram

3.2.7 Opponent Module

Opponents.c

Overall Weight: 8%

Description: This module concerns the game opponents.

The functions present in the module allow to create new opponents, load, draw, erase, move, animate their movement as well as their attack, kill opponents and even detect collisions with the room obstacles or other opponents of the same room.

To save all this information the **Opponent struct** was created. An array of **Opponent** pointers is used to save all the game opponents, and each contains the opponent x and y position, direction, moving animations and attack animation images, speed in both axis and two boolean values that tell if he is alive or not and if he is moving or not.

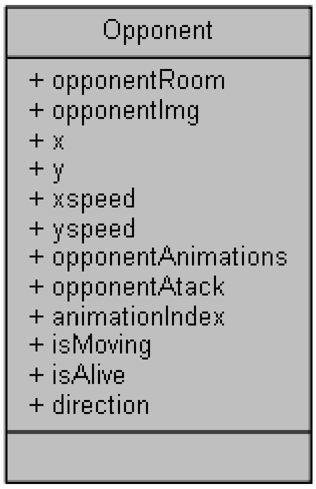


Figure 29 Opponent struct diagram

3.3 Other Modules

3.3.1 Utilities Module

Utils.c

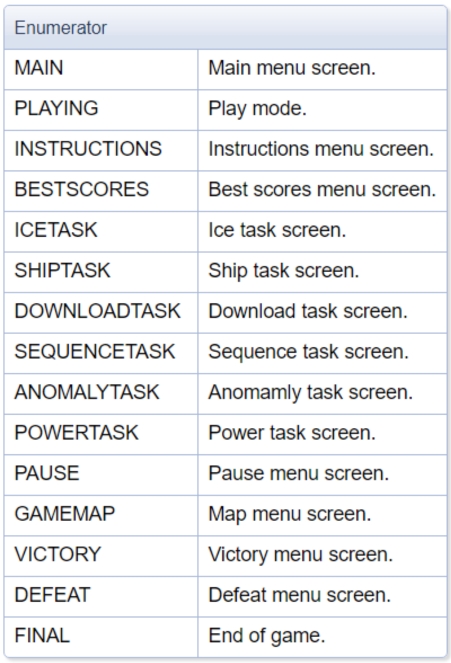
Overall Weight: 1%

Description: In this module are present some functions that help with register manipulation. Its implementation was done during the lab2 week.

4. Implementation details

4.1 State Machines

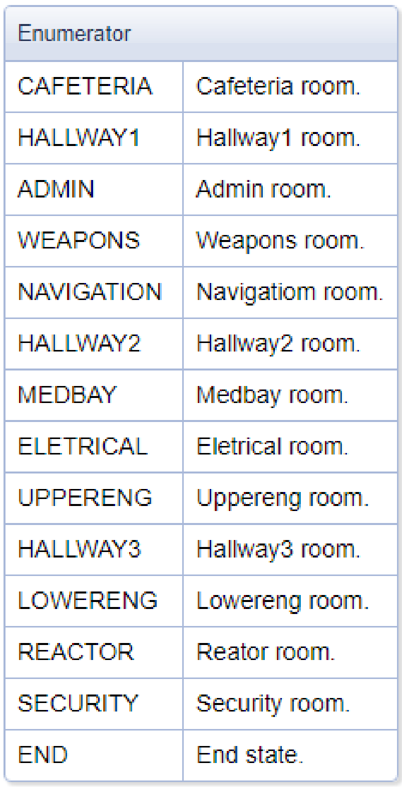
A state machine was implemented, **Menu enum**, that tells in which state the program is currently at. This helps to divide the interrupt handlers and call them according to the **Menu**.



Enumerator	
MAIN	Main menu screen.
PLAYING	Play mode.
INSTRUCTIONS	Instructions menu screen.
BESTSCORES	Best scores menu screen.
ICETASK	Ice task screen.
SHIPTASK	Ship task screen.
DOWNLOADTASK	Download task screen.
SEQUENCETASK	Sequence task screen.
ANOMALYTASK	Anomamly task screen.
POWERTASK	Power task screen.
PAUSE	Pause menu screen.
GAMEMAP	Map menu screen.
VICTORY	Victory menu screen.
DEFEAT	Defeat menu screen.
FINAL	End of game.

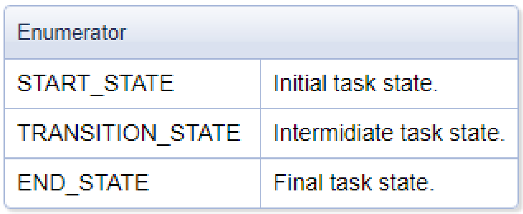
Figura 31 Menu enum diagram

Additionally, another state machine created was the **Room_number**, this tells which is the current room in the PLAYING mode and helps with the room transitions as well as showing all the tasks and opponents of that room.



Enumerator	
CAFETERIA	Cafeteria room.
HALLWAY1	Hallway1 room.
ADMIN	Admin room.
WEAPONS	Weapons room.
NAVIGATION	Navigatiom room.
HALLWAY2	Hallway2 room.
MEDBAY	Medbay room.
ELETRICAL	Eletrical room.
UPPERENG	Uppereng room.
HALLWAY3	Hallway3 room.
LOWERENG	Lowereng room.
REACTOR	Reator room.
SECURITY	Security room.
END	End state.

Figure 31 Room_number enum diagram



Enumerator	
START_STATE	Initial task state.
TRANSITION_STATE	Intermidiate task state.
END_STATE	Final task state.

Figure 32 Task_state enum diagram

A state machine with narrower use was also implemented for the execution of tasks. The **Task_state** allows different tasks to use the same state machine, although having different purposes.

4.2 Partial draws

In this project, most of the draws done are partial, that is when new things are added to the screen, only the new objects xpms are traversed which means a more efficient program.

With this in mind and the cursor object that has free range of movement, a more careful implementation had to be done because the cursor could erase objects that were drawn (not included in the background) if hovered over them. This is the case of the buttons in menus and the game countdown time in the task menu.

4.2 Object-Oriented Programming

Many are the different objects included in the program, from the Player, the Opponent and the Projectiles to the game timer and the different Tasks and Rooms. All of these topics were divided according to their needs, making the code more structured and easier to work with.

Some of these objects have a **Direction** parameter, this is used to know their current direction as well as the way of drawing them. Since including xpms for all animations and directions would be very heavy for the program, it was decided just to include the ones facing right and when the direction was different the way of drawing the object would be different as well.

As for the collisions between these structures, two ways of collision detection were implemented. Between the room and the Player, Opponents and Projectiles, a very practical way was created with the help of a xpm with colour (not transparent) in the places the obstacles were located and not coloured (transparent) in the free moving zone, therefore when checking pixel by pixel if the position of the object in the xpm was not transparent that would mean a collision was occurring.

The second way of checking collisions is a more physical one, it means the collision is detected according to the position of the objects whether it is between opponent/opponent,

opponent/player, opponent/projectile, and the cursor with the buttons. As for the last one, there is a peculiarity to it. In order to be more precise with button collision, in some cases such as the ICE task where the buttons have a hexagonal shape, the values of the position of the cursor would be used and compared with the equation of the limiting lines slope allowing knowing if the cursor would be inside the button or not.

4.3 Event Driven code

All tasks are managed by mouse events, despite having one that is more complex than others, this is the SHIP task. This task has similarities with the function proposed to us in the lab4 week, (mouse_test_gesture) were the mouse had to do an inverted V shape, here the mouse has to create a certain path shown in the task menu while having the left button pressed. This was possible with the help of the state machine stated before (**Task_state**).

4.4 Allocation of memory

Many are the objects that require allocation of memory in the project, consequently when the program ends their space is freed in the gameLoop function, making it a good programming practice.

5. Conclusions

This project started as being very frightening to do, though with the passing of the time it became remarkably interesting and captivating.

Being alone in the making of the project was a bit stressful and not every idea could be included (serial port, harder tasks, and more engagement with the player) with the shortage of time and the existence of other course unit projects. However, the final product was a very remarkable one, in my view, and all the things learned throughout classes were implemented.

A least positive side was the fact that we were not given any experience with the RTC and the Serial port during the practical classes, this led to making it discouraging to use them, because of the lack of information we had about these topics.

To sum up, the Lcom curricular unit is a particularly challenging one having to spend a lot of extra-class time in the studying of its subjects, though it develops our autonomous skills and being able to persist when difficulties come in our way.